JS While Loops

CS 115 Computing for the Socio-Techno Web

Instructor: Brian Brubach

Announcements

- Fill out guest speaker poll
- Social implications Thurs
- PM2 Thurs
- Project meetings Fri
- Assignment 4 out

while statement

- while statement → Control statement which allows JavaScript to repeat a set of statements
- Basic Form

```
while (expression) {
      statements                    // executed as long as expression is true
}
```

- { } are not required if you need to execute only one statement
- You can have other types of statements (including whiles) in a while
- Example file → sqrt-table.html

do while statement

- Executes the statement at least once
- Basic Form

```
do {
    statements                    // executed as long as expression is true
} while (expression);
```

- Notice the semicolon after the expression parenthesis

Combinations of statements

- Keep in mind that you can have any combination of conditionals and iteration (while) statements
- For example:
    - Conditionals insides of loops
    - Conditionals inside conditionals
    - Loops inside conditionals
    - Loops inside of loops

Alert

- We can use the alert function to display information and for debugging purposes
- Notice it prints HTML tags
- How can we do <br>?
  - Use an escape character \n
- What about variables?

```
var x = 3;
alert(“x = ” + x);     // Prints “x = 3”
```

Increment/decrement operators

- ++ → increases value by one
  - x++ is the same as x = x + 1
- -- → decreases value by one
  - x-- is the same as x = x – 1

Assignment operators

- +=
  - x += y is same as x = x + y
- -=
- *=
- /=
- %=
  - % is modulo operator, gives remainder from division
  - 13 % 5 = 3

Infinite loops

- Infinite loop → the expression controlling the loop never becomes false
- Example 1 →

```
int x = 30;
while(x > 0)
        document.writeln("<li>Element</li>");
```

- Example 2 →

```
int x = 7;                  // how about x = 8
while (x != 0)  {
        document.writeln("<li>Element</li>");
        x = x – 2;      // or x -= 2;
}
```

Trace tables

- Mechanism to keep track of values in a program
- Allows you to understand the program behavior
- Useful for learning algorithms or debugging your code
- We could create a trace table for sqrt_table.js

Trace table for sqrt-table.js on input “3”

Current Value	Max Value
0	3
1	3
2	3
3	3
4	3

Designing using pseudocode

- So far we have focused on the syntax and semantics
- As the complexity of problems increases you need a design strategy (algorithm) to solve such problems
- Several alternatives exist to come up with a solution to a problem. A popular one is Pseudocode.
- Pseudocode → English-like description of the set of steps required to solve a problem.
- When you write pseudocode you focus on determining the steps necessary to solve a problem without worrying about programming language syntax issues

In-class draft of pseudocode for finding the minimum value input

```
minVal ← Ask for number
num
do
    num ← Ask for next number or “end”
    if num doesn’t equal ”end” AND num < minVal
        minVal ← num
While num doesn’t equal “end”

Print minVal
```

Solving problems using a programming language

- Pseudocode → Make sure you have written pseudocode
    - Try to verify (e.g., trace tables) that your pseudocode is correct
- Do not wait until the last minute → Code implementation could be unpredictable
- Incremental code development → Fundamental principle in computer programming
    - Write a little bit of code and make sure it works before you move forward
- Don’t make assumptions → If you are not clear about a language construct, write a little program to familiarize yourself with the construct
- Good Indentation → From the get-go use good indentation as it will allow you to understand your code better

Solving problems using a programming language

- Good variable names → Use good variable names from the get-go
- Testing → Test your code with simple cases first
- Keep backups → As you make significant progress in your development, make the appropriate backups
- Trace your code
- Use a debugger
- Take breaks → If you cannot find a bug take a break and come back later
- Comments → Clarify anything that might unclear to someone reading your code (including future you)